Australian Government

Department of Defence

Defence Science and
Technology Organisation

Z Support in the HiVe Mathematical Toolkit

Brendan Mahony, Jim McCarthy, Linh Vu, Kylie Williams

Command, Control, Communications and Intelligence Division

Defence Science and Technology Organisation

DSTO–TR–2272

ABSTRACT

The HiVe project is an ambitious research programme aimed at providing DSTO and the Australian Defence Department with the world's most advanced assurance tools. A key part of this is the provision of advanced high assurance analysis tools in the form of the HiVe Modeller component.

Formal specification and system modelling activities in the HiVe Modeller are supported through an Isabelle/HOL implementation of the HiVe Mathematical Toolkit. This report describes support for the Z Mathematical Toolkit within the HiVe Mathematical Toolkit.

Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone: *(08) 8259 5555*
Facsimile: *(08) 8259 6567*

© Commonwealth of Australia 2009
AR No. AR–014–433
March, 2009

Authors

Dr Brendan Mahony
Defence Science and Technology Organisation

Dr Mahony was admitted as a PhD by the University of Queensland in 1991. After post-doctoral research in high-assurance real-time systems, he joined the DSTO in 1995. With DSTO he has continued his research into high-assurance design methods and has guided the development of the high-assurance tools DOVE and the HiVe.

Jim McCarthy
Defence Science and Technology Organisation

Jim came to the Defence Science and Technology Organisation in 1998 via a career in theoretical/mathematical physics dating back to his Ph.D. from Rockefeller University in 1985. He is currently working to develop high assurance methods and tools, and to model specific (typically infosec) critical systems.

Linh Vu
Defence Sciene and Technology Organisation

Linh Vu has a Bachelor of Science (Mathematical & Computer Sciences), and a Bachelor of Engineering (Computer Systems) from the University of Adelaide. She currently dreams in various programming languages and natural languages in her sleep, and hopes to be a future user of the HiVe.

Ms Kylie Williams
Defence Science and Technology Organisation

Ms Williams received a BCSc in 2000 and a BSc with Honours in Pure Mathematics and Computer Science from the University of Adelaide in 2001. Ms Williams has been with DSTO since 2005 and is currently working with the High Assurance Systems Cell.

Contents

Chapter 1

Introduction

1.1 The Z Toolkit

The Z specification language [2] is widely known and well respected in the formal methods community. Central to the utility of Z is the provision of a standard Mathematical Toolkit for modelling and explaining common problems in Computer Science. So as to maximise accessiblity for this Z community, the HiVe project has undertaken the formalisation of a super-set of the Z Mathematical Toolkit in the Isabelle/HOL theorem proving environment.

Although the Mathematical Toolkit is canonically defined in the ISO Z Standard [2], significant communities exist that observe the definitions provided by Hayes [1] and Spivey [5]. In order to provide the widest possible support to the Z community, the HiVe Mathematical Toolkit includes coverage of all three sources.

This paper describes the Z compatibility features of the HiVe Mathematical Toolkit. Its intended audience is primarily the Z practitioner wishing to make use of the HiVe Mathematical Toolkit with minimal knowledge of the underlying Isabelle environment. A more complete and Isabelle-oriented development of the HiVe Mathematical Toolkit is is given in a separate paper [3].

The main body of the paper is devoted to discussing the HiVe approach to providing Z Mathematical Toolkit support in the Isabelle/HOL environment. The appendices consist of a series of reference pages, in the style of Spivey [5], describing the supported features of the Z Mathematical Toolkit.

1.2 Isabelle/HOL in brief

Isabelle/HOL [4] provides a λ-calculus based modelling and reasoning environment, primarily aimed at those familiar with the concepts and notations of Functional Programming.

The basic term constructors of Isabelle/HOL are

- free variables[1] (eg. x, y, ...)
- constants (eg. $(op =)$, $\{\}$, ...)

[1]For technical reasons there is a second class of *logical* variables, but we ignore this complication here.

- functional abstraction (eg. $(\% \ x. \ x = y)$[2])
- function application (eg. $f\ x\ y$, $(\% \ x. \ x = y)\ 4, \ldots$))

Isabelle/HOL offers basic support for higher-order modelling and reasoning. Firstly, functions are first class objects so that any term, including free and bound variables, may be of function type. Secondly, Isabelle/HOL supports free type variables, allowing the definition of type generic terms. Thirdly, Isabelle/HOL supports type classes (types of types) for restricting the range of type variables to types with common properties (eg. those with well-defined orders). Isabelle/HOL lacks support for more advanced higher-order concepts such as type constructors as first class objects, term-dependent types, or existential types.

The type constructors of Isabelle/HOL are

- free type variables[3] (eg $\alpha, \beta, \ldots$)
- class-constrained variables (eg. $(\alpha{::}order)$, $(\alpha{::}\{order, plus\})$, ...)
- type constructions (eg. nat, $(\alpha,\beta)fun$, $\alpha\ set$, ...)

By convention, modelling in Isabelle/HOL proceeds in a naive declarative style: constants are declared and defined in terms of existing constants, then lemmas and theorems about them are proved. For example:

consts
  my_identity :: (*α*, *α*) *fun*

defs
  my_identity_def: *my_identity* == (% *x*. *x*)

lemma
  my_identity x = *x*
  by (*simp add*: *my_identity_def*)

Declared constants may also be provided with sophisticated mathematical presentations using the **syntax** command.

syntax (*xsymbols*)
  my_identity :: (*α*, *α*) *fun*   (ι _ [*1000*] *999*)

lemma
  ι(ι *x*) = *x*
  by (*simp add*: *my_identity_def*)

Here the token *xsymbols* identifies the *print mode* for the declared syntax and controls when the Isabelle system uses this syntax in its output. The actual syntax is declared at the right end of the declaration: the _ character is the placeholder for the operator argument; the [*1000*] parameter declares the argument priority; and the *999* parameter declares the result priority.

[2]We adopt the basic ascii syntax for HOL throughout, so as to reduce confusion with similar Z notation

[3]Again we ignore the complication of *logical* type variables.

Chapter 2

The Z Expression Language

2.1 Z Expressions as HOL Terms

In modelling the Z expression language, we are faced with the usual questions of deep versus shallow embeddings and how deep to go. We have little interest in being able to reason about the Z expression language as an entity and considerable interest in being able to augment it with the modelling capabilities of Isabelle/HOL. Therefore a full deep embedding is undesirable. In fact we see considerable benefits in making the model as shallow as possible, including actually adopting existing HOL features where the corresponding Z feature is similar in intent.

The Z expression language is split strongly between predicates and (non-predicate) expressions.

The predicate laguage is essentially identical in intent to the HOL boolean type and its associated algebra. We simply replace the predicate language with the terms of the boolean type. Similarly, we identify the expression language with the terms of the respective HOL types.

As noted above, HOL provides implementations of all the basic Z type constructors: sets, cross products, numbers, sequences, and bags.

For sets and cross products, we see no practical distinction between HOL and Z, so we simply adopt the HOL model for Z.

For sequences (called *lists* in HOL) and bags (*multisets* in HOL) the differences are of a fundamental nature. In particular, Z sequences and bags are graphs and users often make use of graph operators in dealing with them. Consequently, we felt it imperative to provide first-class Z implementations of sequences and bags.

In the case of numbers, the correct path was not so clear cut. HOL provides distinct types of naturals, integers and reals, whereas Z provides only the abstract type of *arithos*, with naturals, integers and reals (if adopted) as subsets of arithos. The Z approach offers some convenience in avoiding the use of type coercions, but providing a separate implementation of arithmetic would be prohibitively expensive. Fortunately, the majority of HOL's development of arithmetic is type generic in nature and we are readily able to introduce the arithos type, while retaining the extensive HOL development. Currently, this type is instantiated to the reals, but a larger arithmetic domain could readily be adopted.

Most of this machinery of HOL is familar to the Z practioner, but some of the syntactic sugar is not and is likely to be annoying. For example, the standard binder separator is the weak and easily missed fullstop character, eg. ($\%\ x.\ f\ x$) rather than ($\lambda\ x \bullet f\ x$). In order to improve readibility for the Z practioner, we introduce an Isabelle print mode *zed* with associated Z-ified syntactic operators that support the basic syntax of Z.

Presenting the definition of this *zed* syntax to the reader familiar with Z presents something of a difficulty. The Isabelle mechanisms for decribing syntax are very powerful, but not really very accessible, even for the Isabelle specialist. Besides, while it is easy to quote the text of theorems in general and definitions in particular, the Isabelle tool doesn't offer any satisfactory mechanism for automatically quoting the text of a constant or syntax declarations. Given that we are forced to some form of paraphrase of the declarations, we adopt a convention of presenting them in the form of term definitions. Such mock declarations are at least easily comprehended, if not entirely satisfactory in the formal sense.

The syntax and constants required to support the basic Z expression language are presented in the following sections in this fashion.

2.2 Predicates

This section contains descriptions of the basic predicate operators:

$=$, $\in$	Equality, set membership (p. 6)
true, false	Boolean values (p. 7)
$\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$	Propositional connectives (p. 8)
$\forall$, $\exists$, $\exists_1$	Quantifiers (p. 9)

Name

$=$	-	Equality
$\in$	-	Set membership

Definition

$x = y == x = y$
$x \in X == x : X$

Description

Equality and set membership are boolean-valued binary operators and form part of the HOL term algebra.

Laws

$$x = x \qquad (\mathit{refl})$$
$$x = y \Rightarrow y = x \qquad (\mathit{Z\_sym})$$
$$x = y \land y = z \Rightarrow x = z \qquad (\mathit{Z\_trans})$$
$$S = T \Leftrightarrow (\forall\ x \bullet x \in S \Leftrightarrow x \in T) \qquad (\mathit{Z\_seq\_eq\_def})$$
$$x = y \Leftrightarrow (\forall\ S \bullet x \in S \Leftrightarrow y \in S) \qquad (\mathit{Z\_member\_congruence})$$

Name

true	-	Truth
false	-	Falsity

Definition

true == *True*
false == *False*

Description

The predicates true, false are equated to the corresponding boolean operators.

Laws

false ≠ true	(*False_not_True*)
$((P \Leftrightarrow \text{true}) \Rightarrow R) \land ((P \Leftrightarrow \text{false}) \Rightarrow R) \Rightarrow R$	(*Z_bool_cases*)
$\text{false} \Rightarrow P$	(*Z_FalseE*)
$P \Rightarrow \text{true}$	(*Z_TrueI*)

Name

$\neg$	-	Negation
$\wedge$	-	Conjunction
$\vee$	-	Disjunction
$\Rightarrow$	-	Implication
$\Leftrightarrow$	-	Equivalence

Definition

$\neg\ A == \tilde{}\ A$
$A \wedge B == A\ \&\ B$
$A \vee B == A \mid B$
$A \Rightarrow B == A\ {-}{-}{>}\ B$
$A \Leftrightarrow B == A = B$

Description

The Z predicate connectives are equated to the corresponding HOL operators.

Laws

$(P \Rightarrow \text{false}) \Rightarrow \neg\ P$ (*Z_notI*)
$P \Rightarrow Q \Rightarrow P \wedge Q$ (*Z_conjI*)
$(P \Rightarrow P \vee Q) \wedge (Q \Rightarrow P \vee Q)$ (*Z_disjI*)
$(P \Rightarrow Q) \wedge (Q \Rightarrow P) \Rightarrow (P \Leftrightarrow Q)$ (*Z_iffI*)

Name

$\forall$	-	Universal quantifier
$\exists$	-	Existential quantifier
$\exists_1$	-	Unique quantifier

Definition

$(\forall\ x \mid Q\,x \bullet P\,x) == (!\ x.\ Q\,x \dashrightarrow P\,x)$
$(\exists\ x \mid Q\,x \bullet P\,x) == (?\,x.\ Q\,x\ \&\ P\,x)$
$(\exists_1\ x \mid Q\,x \bullet P\,x) == (?!\ x.\ Q\,x\ \&\ P\,x)$

Description

We model the boolean quantifiers as higher-order operators, taking a boolean valued operator and returning a boolean value. This is a significant difference from the Z approach of schema-text local variables, but gives the same high level reasoning rules and allows full utilisation of Isabelle's efficient treatment of bound variables.

Laws

$(\forall\ x \bullet P\,x) \Leftrightarrow \neg\,(\exists\ x \bullet \neg\,P\,x)$ (*all_conv*)
$(\exists\ x \bullet P\,x) \Leftrightarrow \neg\,(\forall\ x \bullet \neg\,P\,x)$ (*ex_conv*)
$(\forall\ x \bullet x = v\ \Rightarrow P\,x) \Leftrightarrow P\,v$ (*one_point_all*)
$(\exists\ x \bullet x = v \wedge P\,x) \Leftrightarrow P\,v$ (*one_point_ex*)
$(\exists_1\ x \bullet P\,x) \Leftrightarrow (\exists\ x \bullet P\,x \wedge (\forall\ y \bullet P\,y\ \Rightarrow y = x))$ (*Z_ex1_def*)

2.3 Expressions

This section contains descriptions of the basic expression operators:

$(\ldots), \{\ldots\}$	Tuple and set extension (p. 11)
$\mathbb{P}, \times$	Power set and cartesian product (p. 12)
$\{ \mid \bullet \}$	Set comprehension (p. 13)
λ, μ	Lambda-expression and unique choice (p. 14)
let	Let-expression (p. 15)
$\cdot$	(Graph) Function application (p. 16)
if then	Condition expression (p. 17)

Name

$(\ldots)$	-	Tuple
$\{\ldots\}$	-	Set extension

Notation

We write $(x_0, x_1, \ldots, x_n)$ for the tuple *Pair* x_0 $(x_1, \ldots x_n)$.
We write $\{x_0, x_1, \ldots, x_n\}$ for the set *insert* x_0 $\{x_1, \ldots x_n\}$.

Description

Tuple and set extension are syntactic sugar for the Isabelle/HOL *Pair* and *insert* operators respectively.

Laws

$(a, b) = (a', b') \Leftrightarrow a = a' \wedge b = b'$ (*Pair_eq*)
$a \in \{\mathrm{b}\} \cup \mathrm{A} \Leftrightarrow a = b \vee a \in A$ (*insert_iff*)

Name

$\mathbb{P}$	-	Power set
$\times$	-	Cartesian product

Definition

$\mathbb{P}\, X == \mathit{Pow}\, X$
$X \times Y == X <\!*\!> X$

Description

Power set and cross product are modelled in Isabelle/HOL as set operators.

Laws

$$X \times Y = \{\ x\ y \mid x \in X \wedge y \in Y \bullet (x, y)\ \} \qquad (\mathit{Z\_prod\_def})$$

Name

$\{ \mid \bullet \}$ - Set comprehension

Definition

$\{ x \mid P\, x \bullet t\, x \} == \{ t\, x \mid x.\ P\, x \}$

Description

Set comprehension is modelled in Isabelle/HOL as a function from boolean operators to sets. This provides the essential properties of the Z set comprehension, though as usual the bound variable modelling differs.

Laws

$$y \in \{ x \mid Q\, x \bullet t\, x \} \Leftrightarrow (\exists\ x \bullet Q\, x \wedge y = t\, x) \qquad (\mathit{Z\_coll\_mem})$$

Name

λ	-	Lambda-expression
μ	-	Unique choice

Definition

$(\lambda\ x \mid Q\ x \bullet t\ x) == \{\ x \mid Q\ x \bullet (x \mapsto t\ x)\ \}$
$(\mu\ x \mid Q\ x \bullet t\ x\ ) == \mathit{The}\ (\%\ y.\ y : \{\ t\ x \mid x.\ Q\ x\ \})$

Description

The (graph) lambda-expression is not modelled in Isabelle/HOL. We define it as a graph-valued operator with two arguments, the domain term and the result term.

The unique choice operator is modelled in Isabelle/HOL.

Laws

$(u \mapsto v) \in (\lambda\ x \mid b\ x \bullet t\ x) \Leftrightarrow b\ u \wedge v = t\ u$	(*Z_glambda_mem*)
$b\ e\ \Rightarrow (\lambda\ x \mid b\ x \bullet t\ x) \cdot e = t\ e$	(*Z_glambda_beta*)
$\mathrm{dom}\ (\lambda\ x \mid b\ x \bullet t\ x) = \{\ x \mid b\ x\ \}$	(*glambda_dom*)
$\mathrm{ran}\ (\lambda\ x \mid b\ x \bullet t\ x) = \{\ x \mid b\ x \bullet t\ x\ \}$	(*glambda_ran*)
$P\ a\ \Rightarrow (\forall\ x \bullet P\ x\ \Rightarrow x = a)\ \Rightarrow (\mu\ x \mid P\ x) = a$	(*Z_collect_the_equality*)

Name

let - Local definition

Definition

let $x = t \bullet f\,x$ end == (% $x.\ f\,x$) t

Description

The let expression is modelled in Isabelle as an operator that acts on a term and a function.

Name

$\cdot$ - (Graph) Function application

Definition

$f \cdot x == \mathit{The}\ (\%\ y.\ (x, y) \in f)$

Description

(Graph) Function application is not modelled in Isabelle/HOL. We define it as an operator with two arguments, the graph and the argument.

Laws

$(\exists_1\ y \bullet (x \mapsto y) \in f) \Rightarrow (x \mapsto f \cdot x) \in f$ (*Z_single_val_appl*)
$(x \mapsto y) \in f \Rightarrow f \cdot x = y$ (*Z_pfun_beta*)
$x \in \mathrm{dom}\ f \Rightarrow (x \mapsto f \cdot x) \in f$ (*Z_pfun_appl*)
$x \in \mathrm{dom}\ f \Rightarrow ((x \mapsto y) \in f \Leftrightarrow y = f \cdot x)$ (*Z_pfun_unique*)

Name

if then - Conditional expression

Definition

if b then u else v fi == *if b then u else v*

Description

The condition expression is defined in Isabelle/HOL as a three-place operator.

Laws

$P \Rightarrow$ if P then E_1 else E_2 fi = E_1 (Z_true_if)
$\neg P \Rightarrow$ if P then E_1 else E_2 fi = E_2 (Z_false_if)
if P then E else E fi = E (Z_idem_if)

Chapter 3

The Z Mathematical Toolkit

3.1 Modelling issues in Isabelle/HOL

The primary issue in modelling the Z Mathematical Toolkit in Isabelle/HOL lies in Z's use of sets of pairs to model all functions. HOL provides a built-in type construction of total functions that is distinct from the graph type. For the purposes of clarity, we refer to these HOL functions as *operators* and Z functions as *graphs*.

Clearly the graph fills such a central role in Z and provides such a flexible mechanism for finite data structures that any implementation of the Z Toolkit must provide full first-class support for graphs. On the other hand, for many algebraic primitives, the operator offers significant advantages, reducing syntactic baggage and eliminating the need to reason about definedness. Besides, rejecting the use of operators completely would deny access to the large suite of modelling tools defined in HOL. It is clear that the HiVe user will be best suited by having access to both worlds.

Having decided to proceed with full support for both function models in the HiVe, one is left with the tricky decision of how much to make use of operators in supporting the Z Toolkit. A totally pure approach of not allowing operator models for any Z constructs would be expensive and brittle. For example, requiring a complete re-implementation of arithmetic! In any case, the use of graphs to model what are essentially algebraic operators is often awkward and unsatisfying in the Z Standard [2]. Our approach has been to use the operator model where a construct is basically an algebraic operator and the graph model where it is to be used primarily for user-level modelling.

In Z, the only mechanism for genericity is the given type. Again this is often awkward, as seen in the convention of leaving out generic parameters in most cases. Generally, HOL-style type generics offer a better solution to type abstraction. Our approach is to use type generics wherever possible, adopting set generics only where the value of the set parameter actually changes the meaning of the object. Where we use set generics, we model it as a set-valued argument to the constant.

This leads nicely to discussion of another fundamental question. Whether to introduce Z constructs as HOL constants or else to adopt an explicit model of the environment in the semantics presented in the Z Standard [2]. Thus far we can see no compelling argument for pursuing the latter option and a number of barriers, such as developing an appropriate data type for modelling such an environment. All the standard elements of the Z Toolkit are introduced as constants as described in Section 1.2.

Finally, as discussed in Section 2.1, we note that HOL provides implementations of all the basic Z type constructors, sets, cross products, numbers, sequences, and bags.

For sets and cross products, we see no fundamental distinction between HOL and Z, so we simply adopt the HOL model for Z.

For sequences (called *lists* in HOL) and bags (*multisets* in HOL) the differences are of a fundamental nature. In particular, Z sequences and bags are graphs and users often make use of graph operators in dealing with them. Consequently, we felt it imperative to provide first-class Z implementations of sequences and bags.

In the case of numbers, the correct path was not so clear cut. HOL provides distinct types of naturals, integers and reals, whereas Z provides only the abstract type of *arithos*, with naturals, integers and reals (if adopted) as subsets of arithos. The Z approach offers some convenience in avoiding the use type coercions, but providing a separate implementation of arithmetic would be prohibitively expensive. Fortunately, the majority of HOL's development of arithmetic is type generic in nature and we were readily able to define the arithos type, while retaining the extensive HOL development.

3.2 Sets

Name

$\neq$	-	Inequality
$\notin$	-	Non-membership

Definition

$x \neq y \equiv \neg\,(x = y)$ $\qquad(Z\_neq\_def)$

$x \notin S \equiv \neg\,(x \in S)$ $\qquad(Z\_nin\_def)$

Description

The negations of equality and set membership [5, p 89][2, p 95] are already defined as syntactic operators in HOL. We simply make use of these existing, type-generic operators.

Laws

$x \neq y \;\Rightarrow y \neq x$ $\qquad(Z\_neq\_commute)$

Name

$\varnothing$	-	Empty set
$\mathfrak{U}$	-	Universal set
$\subseteq$	-	Subset relation
$\subset$	-	Proper subset relation
$\mathbb{P}_1$	-	Non-empty subsets

Definition

$\varnothing \equiv \{\, x \mid \text{false} \,\}$ (*Z_empty_def*)

$\mathfrak{U} \equiv \{\, x \mid \text{true} \,\}$ (*Z_UNIV_def*)

$S \subseteq T \equiv \forall\ x \bullet x \in S\ \Rightarrow x \in T$ (*Z_subseteq_def*)

$S \subset T \equiv S \subseteq T \wedge S \neq T$ (*Z_subset_def*)

$\mathbb{P}_1\ X \equiv \{\, S \mid S \in \mathbb{P}\ X \wedge S \neq \varnothing \,\}$ (*Z_Pow1_def*)

Description

The empty set and subset relations [5, p 90][2, p 95] are defined as operators in HOL. We make use of the existing, type-generic operators, with appropriate Z-style syntax.

The non-empty power set [5, p 90][2, p 96] we define as an operator on sets.

Laws

$x \notin \varnothing$ (*Z_notin_empty*)

$S \subseteq T \Leftrightarrow S \in \mathbb{P}\ T$ (*Z_subset_Pow*)

$S \subseteq S$ (*Z_subset_refl*)

$\neg\ (S \subset S)$ (*Z_psubset_not_refl*)

$S \subseteq T \wedge T \subseteq S \Leftrightarrow S = T$ (*Z_subset_antisym*)

$\neg\ (S \subset T \wedge T \subset S)$ (*Z_psubset_chained*)

$S \subseteq T \wedge T \subseteq V\ \Rightarrow S \subseteq V$ (*Z_subset_trans*)

$S \subset T \wedge T \subset V\ \Rightarrow S \subset V$ (*Z_psubset_trans*)

$\varnothing \subseteq S$ (*Z_empty_subset*)

$\varnothing \subset S \Leftrightarrow S \neq \varnothing$ (*Z_empty_psubset*)

$\mathbb{P}_1\ X = \varnothing \Leftrightarrow X = \varnothing$ (*Z_Pow1_empty*)

$X \neq \varnothing \Leftrightarrow X \in \mathbb{P}_1\ X$ (*Z_nempty_Pow1*)

Name

$\cup$	-	Set union
$\cap$	-	Set intersection
$\setminus$	-	Set difference

Definition

$S \cap T \equiv \{ x \mid x \in S \land x \in T \}$ (*Z_inter_def*)
$S \cup T \equiv \{ x \mid x \in S \lor x \in T \}$ (*Z_union_def*)
$S \setminus T \equiv \{ x \mid x \in S \land x \notin T \}$ (*Z_set_diff_def*)

Description

Set union, intersection, and difference [5, p 91][2, p 97] are already defined in HOL. We make use of the existing, type-generic operators, with appropriate Z-style syntax. The symmetric set difference operator [2, p 97] is not already defined in HOL. We define it as a binary set operator.

Laws

$S \cup S = S \cup \varnothing = S \cap S = S \setminus \varnothing = S$ (*Z_union_inter_diff_idem*)
$S \cap \varnothing = S \setminus S = \varnothing \setminus S = \varnothing$ (*Z_inter_diff_empty*)
$S \cup T = T \cup S$ (*Z_union_comm*)
$S \cap T = T \cap S$ (*Z_inter_comm*)
$S \cup (T \cup V) = S \cup T \cup V$ (*Z_union_assoc*)
$S \cap (T \cap V) = S \cap T \cap V$ (*Z_inter_assoc*)
$S \cup T \cap V = (S \cup T) \cap (S \cup V)$ (*Z_union_dist*)
$S \cap (T \cup V) = S \cap T \cup S \cap V$ (*Z_inter_dist*)
$S \cap T \cup (S \setminus T) = S$ (*Z_partition*)
$S \cup (T \setminus V) = S \cup T \setminus (V \setminus S)$ (*Z_union_diff*)
$(S \setminus T) \cap T = \varnothing$ (*Z_diff_disjoint*)
$S \cap (T \setminus V) = S \cap T \setminus V$ (*Z_inter_diff*)
$S \setminus (T \setminus V) = (S \setminus T) \cup S \cap V$ (*Z_diff_diff1*)
$S \cup T \setminus V = (S \setminus V) \cup (T \setminus V)$ (*Z_diff_union*)
$S \setminus T \setminus V = S \setminus T \cup V$ (*Z_diff_diff2*)
$S \setminus T \cap V = (S \setminus T) \cup (S \setminus V)$ (*Z_diff_inter*)

Name

$\bigcup$	-	Generalised union
$\bigcap$	-	Generalised intersection

Definition

$\bigcup A \equiv \{x \mid \exists\ S \bullet S \in A \land x \in S\}$ (*Z_Union_def*)

$\bigcap A \equiv \{\ x \mid \forall\ S \bullet S \in A\ \Rightarrow x \in S\ \}$ (*Z_Inter_def*)

Description

Generalised union and intersection [5, p 92][2, p 97] are defined as operators in HOL. We make use of the existing, type-generic operators, with appropriate Z-style syntax. We also allow the dropping of the set brackets when applied to set comprehensions, i.e. $(\bigcup\ x \mid Q\ x \bullet t\ x)$ and $(\bigcap\ x \mid Q\ x \bullet t\ x)$.

Laws

$\bigcup(A \cup B) = \bigcup A \cup \bigcup B$ (*Z_Union_union_dist*)

$\bigcap(A \cup B) = \bigcap A \cap \bigcap B$ (*Z_Inter_union_dist*)

$\bigcup \varnothing = \varnothing$ (*Z_Union_empty*)

$\bigcap \varnothing = \mathfrak{U}$ (*Z_Inter_empty*)

$S \cap \bigcup A = (\bigcup\ T \mid T \in A \bullet S \cap T)$ (*Z_inter_Union_dist*)

$\bigcup A \setminus S = (\bigcup\ T \mid T \in A \bullet T \setminus S)$ (*Z_Union_diff_dist*)

$S \setminus \bigcap A = (\bigcup\ T \mid T \in A \bullet S \setminus T)$ (*Z_diff_Inter_dist*)

$A \neq \varnothing\ \Rightarrow \bigcap A \setminus S = (\bigcap\ T \mid T \in A \bullet T \setminus S)$ (*Z_Inter_diff_dist*)

$A \subseteq B\ \Rightarrow \bigcup A \subseteq \bigcup B$ (*Z_Union_mono*))

$A \subseteq B\ \Rightarrow \bigcap B \subseteq \bigcap A$ (*Z_Inter_antimono*)

Name

fst, snd - Projection functions for ordered pairs

Definition

$\mathrm{fst}\ (x, y) \equiv x$ (Z_fst_def)

$\mathrm{snd}\ (x, y) \equiv y$ (Z_snd_def)

Description

The first and second operators [5, p 93][2, p 98] are defined in HOL. These are not strictly compatible with those defined in the Z Standard, since the Z operators act on bindings rather than tuples, which are subsumed by the binding structure in Z. Nevertheless we find it convenient to make use of the HOL operators. As noted elsewhere, bindings are a difficult structure to model in HOL.

Laws

$(\mathrm{fst}\ p, \mathrm{snd}\ p) = p$ (Z_tuple_cong)

Order properties of set operations

$$S \subseteq S \cup T \tag{\textit{union\_ub1}}$$
$$T \subseteq S \cup T \tag{\textit{union\_ub2}}$$
$$S \subseteq W \land T \subseteq W \;\Rightarrow S \cup T \subseteq W \tag{\textit{union\_least}}$$
$$S \in A \;\Rightarrow S \subseteq \bigcup A \tag{\textit{Union\_ub}}$$
$$(\forall\ S \bullet S \in A \;\Rightarrow S \subseteq W) \;\Rightarrow \bigcup A \subseteq W \tag{\textit{Union\_least}}$$
$$S \cap T \subseteq S \tag{\textit{inter\_lb1}}$$
$$S \cap T \subseteq T \tag{\textit{inter\_lb2}}$$
$$W \subseteq S \land W \subseteq T \;\Rightarrow W \subseteq S \cap T \tag{\textit{inter\_greatest}}$$
$$S \in A \;\Rightarrow \bigcap A \subseteq S \tag{\textit{Z\_Inter\_lb}}$$
$$(\forall\ S \bullet S \in A \;\Rightarrow W \subseteq S) \;\Rightarrow W \subseteq \bigcap A \tag{\textit{Z\_Inter\_greatest}}$$
$$S \setminus T \subseteq S \tag{\textit{diff\_lb}}$$
$$W \subseteq S \land W \cap T = \varnothing \;\Rightarrow W \subseteq S \setminus T \tag{\textit{diff\_greatest}}$$

3.3 Relations

Name

$\leftrightarrow$	-	Binary relation graphs
$\mapsto$	-	Maplet

Definition

$$X \leftrightarrow Y \equiv \mathbb{P}\,(X \times Y) \qquad (\mathit{rel\_def})$$
$$x \mapsto y \equiv (x, y) \qquad (\mathit{Z\_maplet\_def})$$

Description

Isabelle/HOL allows us to adopt the usual Z model of relations as sets of pairs. We call this model the *graph* approach. Another approach would be to model relations as binary boolean-valued operators. Isabelle/HOL makes use of both models in its development, making it necessary to convert between the two at times. We write op r for the operator generated by the graph r and grf s for the graph generated by the operator s.

Following Spivey [5], we adopt syntax for infix relation application (for example writing a $\underline{R}$ b for $(a \mapsto b) \in R$) and relational chaining (for example writing $a, b \in X \subseteq Y$ for $a \in X \land b \in X \land X \subseteq Y$).

HOL provides a built-in model for functions (value abstractions), embodied by the type constructor *fun* and the λ-constructor. In the following we refer to this model as the *operator* model of functions.

The single-valued *graphs* provide a convenient (and widely used) mathematical model of functions. This is especially so when partial or finite functions are of particular interest, as is often the case in program specification. A graph is a set of pairs describing the relationship between function argument and function result.

Name

dom	-	Domain
ran	-	Range

Definition

$\mathrm{dom}\ R \equiv \{\ x\ y \mid x\ \underline{R}\ y \bullet x\ \}$ (*Z_dom_def*)

$\mathrm{ran}\ R \equiv \{\ x\ y \mid x\ \underline{R}\ y \bullet y\ \}$ (*Z_ran_def*)

Description

Domain and range operators [5, p 96][2, p 98] are already defined.

Laws

$x \in \mathrm{dom}\ r \Leftrightarrow (\exists\ y \bullet y \in Y \wedge x\ \underline{r}\ y)$ (*Z_in_domD*)

$y \in \mathrm{ran}\ r \Leftrightarrow (\exists\ x \bullet x \in X \wedge x\ \underline{r}\ y)$ (*Z_in_ranD*)

$\mathrm{dom}\ \{(x, y)\} \cup R = \{x\} \cup \mathrm{dom}\ R$ (*Z_rel_insert_dom*)

$\mathrm{ran}\ \{(x, y)\} \cup R = \{y\} \cup \mathrm{ran}\ R$ (*Z_rel_insert_ran*)

$\mathrm{dom}\ (R_1 \cup R_2) = \mathrm{dom}\ R_1 \cup \mathrm{dom}\ R_2$ (*Z_rel_union_dom*)

$\mathrm{ran}\ (R_1 \cup R_2) = \mathrm{ran}\ R_1 \cup \mathrm{ran}\ R_2$ (*Z_rel_union_ran*)

$\mathrm{dom}\ (R_1 \cap R_2) \subseteq \mathrm{dom}\ R_1 \cap \mathrm{dom}\ R_2$ (*Z_rel_inter_dom*)

$\mathrm{ran}\ (R_1 \cap R_2) \subseteq \mathrm{ran}\ R_1 \cap \mathrm{ran}\ R_2$ (*Z_rel_inter_ran*)

$\mathrm{dom}\ \varnothing = \varnothing$ (*Z_rel_empty_dom*)

$\mathrm{ran}\ \varnothing = \varnothing$ (*Z_rel_empty_ran*)

Name

id	-	Identity relation
$\fatsemi$	-	Relational composition
$\circ$	-	Backward relational composition

Definition

$\mathrm{id}\ X \equiv \{\ x \mid x \in X \bullet (x, x)\ \}$ (*rel_id_def*)

$R \circ Q \equiv \{\ x\ y\ z \mid x\ \underline{Q}\ y \wedge y\ \underline{R}\ z \bullet x \mapsto z\ \}$ (*Z_comp_def*)

$Q \fatsemi R \equiv R \circ Q$ (*Z_fcomp_def*)

Description

Relational identity [5, p 97][2, p 98] and (backward) composition [5, p 97][2, p 99] are already defined in HOL. We introduce forward composition [5, p 97][2, p 99] by identifying it with backward composition, but with the arguments reversed.

Laws

$(x \mapsto x') \in \mathrm{id}\ X \Leftrightarrow x = x' \in X$ (*Z_rel_id_mem*)

$(x \mapsto z) \in P \fatsemi Q \Leftrightarrow (\exists\ y \bullet x\ \underline{P}\ y \wedge y\ \underline{Q}\ z)$ (*Z_rel_fcomp_mem*)

$(x \mapsto z) \in P \circ Q \Leftrightarrow (\exists\ y \bullet x\ \underline{Q}\ y \wedge y\ \underline{P}\ z)$ (*Z_rel_comp_mem*)

$(P \fatsemi Q) \fatsemi R = P \fatsemi Q \fatsemi R$ (*Z_rel_fcomp_assoc*)

$\mathrm{id}\ (\mathrm{dom}\ P) \fatsemi P = P$ (*Z_rel_lident′*)

$P \fatsemi \mathrm{id}\ (\mathrm{ran}\ P) = P$ (*Z_rel_rident′*)

$\mathrm{id}\ V \fatsemi \mathrm{id}\ W = \mathrm{id}\ (V \cap W)$ (*Z_rel_id_fcomp*)

Name

$\lhd$	-	Domain restriction
$\rhd$	-	Range restriction

Definition

$S \lhd R \equiv \{ x\ y \mid x \in S \land x\ \underline{R}\ y \bullet x \mapsto y \}$ (*Z_dres_def*)

$R \rhd T \equiv \{ x\ y \mid y \in T \land x\ \underline{R}\ y \bullet x \mapsto y \}$ (*Z_rres_def*)

Description

Domain and range restrictions [5, p 98][2, p 99] are not defined in HOL. In the Z standard, they are defined in a set generic manner, but they do not vary in value according to the carrier set, so we define them in a type generic manner.

Laws

$S \lhd R = \mathrm{id}\ S \mathbin{;} R = S \times Y \cap R$ (*Z_dres_id_inter*)

$R \rhd T = R \mathbin{;} \mathrm{id}\ T = R \cap X \times T$ (*Z_rres_id_inter*)

$\mathrm{dom}\ (U \lhd R) = U \cap \mathrm{dom}\ R$ (*Z_dres_dom*)

$\mathrm{ran}\ (R \rhd T) = \mathrm{ran}\ R \cap T$ (*Z_rres_ran*)

$S \lhd R \subseteq R$ (*Z_dres_sub_self*)

$R \rhd T \subseteq R$ (*Z_rres_sub_self*)

$U \lhd R \rhd V = U \lhd (R \rhd V)$ (*Z_dr_res_assoc*)

$U \lhd V \lhd R = (U \cap V) \lhd R$ (*Z_dres_dist*)

$R \rhd U \rhd V = R \rhd (U \cap V)$ (*Z_rres_dist*)

Name

$\ntriangleleft$	-	Domain anti-restriction
$\ntriangleright$	-	Range anti-restriction

Definition

$S \ntriangleleft R \equiv \{\, x\ y \mid x \notin S \wedge x\ \underline{R}\ y \bullet x \mapsto y \,\}$ $\qquad (Z\_dsub\_def)$

$R \ntriangleright T \equiv \{\, x\ y \mid y \notin T \wedge x\ \underline{R}\ y \bullet x \mapsto y \,\}$ $\qquad (Z\_rsub\_def)$

Description

As above, domain and range antirestrictions [5, p 99][2, p 99,100] are not a standard part of HOL. In the Z standard, they are defined in a set generic manner, but they do not vary in value according to the carrier set, so we define them in a type generic manner.

Laws

$U \ntriangleleft R = (\mathfrak{U} \setminus U) \triangleleft R$ $\qquad (Z\_dsub\_id\_char)$

$R \ntriangleright V = R \triangleright (\mathfrak{U} \setminus V)$ $\qquad (Z\_rsub\_id\_char)$

$U \triangleleft R \cup U \ntriangleleft R = R$ $\qquad (Z\_dpart\_rel)$

$R \triangleright T \cup R \ntriangleright T = R$ $\qquad (Z\_rpart\_rel)$

Name

$^\sim$ - Relational inverse

Definition

$R^\sim \equiv \{\, x\ y \mid x\ \underline{R}\ y \bullet y \mapsto x \,\}$ (*Z_inverse_def*)

Description

The relational inverse [5, p 100][2, p 100] is already defined in HOL.

Laws

$(x \mapsto y) \in R^\sim \Leftrightarrow (y \mapsto x) \in R$ (*Z_inverse_mem*)

$(R^\sim)^\sim = R$ (*Z_inverse_idem*)

$(R \circ S)^\sim = S^\sim \circ R^\sim$ (*Z_inverse_rel_comp*)

$(\mathrm{id}\ X)^\sim = \mathrm{id}\ X$ (*Z_inverse_id*)

$\mathrm{dom}\ (R^\sim) = \mathrm{ran}\ R$ (*Z_inverse_dom*)

$\mathrm{ran}\ (R^\sim) = \mathrm{dom}\ R$ (*Z_inverse_ran*)

$\mathrm{id}\ (\mathrm{dom}\ R) \subseteq R \fatsemi R^\sim$ (*Z_inverse_lgalois*)

$\mathrm{id}\ (\mathrm{ran}\ R) \subseteq R^\sim \fatsemi R$ (*Z_inverse_rgalois*)

Name

$\_ (\!| \_ |\!)$ - Relational image

Definition

$R(\!|S|\!) \equiv \{\, x\ y \mid x \in S \wedge x\ \underline{R}\ y \bullet y \,\}$ (*Z_Image_def*)

Description

Again relational image [5, p 101][2, p 100] is a standard part of HOL.

Laws

$y \in R(\!|U|\!) \Leftrightarrow (\exists\ x \bullet x \in U \wedge x\ \underline{R}\ y)$ (*Z_Image_diff*)
$R(\!|U|\!) = \mathrm{ran}\,(U \lhd R)$ (*Z_Image_dres*)
$\mathrm{dom}\,(S \mathbin{;} R) = (S^{\sim})(\!|\mathrm{dom}\ R|\!)$ (*Z_inv_Image_dom_rel*)
$\mathrm{ran}\,(S \mathbin{;} R) = R(\!|\mathrm{ran}\ S|\!)$ (*Z_Image_ran_rel*)
$R(\!|U \cup V|\!) = R(\!|U|\!) \cup R(\!|V|\!)$ (*Z_Image_union*)
$R(\!|U \cap V|\!) \subseteq R(\!|U|\!) \cap R(\!|V|\!)$ (*Z_Image_inter*)
$R(\!|\mathrm{dom}\ R|\!) = \mathrm{ran}\ R$ (*Z_Image_dom*)
$\mathrm{dom}\ R = \mathrm{fst}(\!|R|\!)$ (*Z_dom_Image*)
$\mathrm{ran}\ R = \mathrm{snd}(\!|R|\!)$ (*Z_ran_Image*)

Name

$\oplus$ - Overriding

Definition

$Q \oplus R \equiv \mathrm{dom}\, R \mathbin{⩤} Q \cup R$ ($Z\_rel\_oride\_def$)

Description

Relational overriding [5, p 102][2, p 100] is not a standard part of HOL. As its value does not vary with carrier set, we make a type generic definition.

Laws

$R \oplus R = R$ ($Z\_rel\_oride\_idem$)
$(P \oplus Q) \oplus R = P \oplus Q \oplus R$ ($Z\_rel\_oride\_assoc$)
$\varnothing \oplus R = R \oplus \varnothing = R$ ($Z\_rel\_oride\_id$)
$\mathrm{dom}\,(Q \oplus R) = \mathrm{dom}\, Q \cup \mathrm{dom}\, R$ ($Z\_rel\_oride\_dom\_dist$)
$Q \oplus R = Q \cup R$ ($Z\_rel\_oride\_disj$)
$V \lhd (Q \oplus R) = V \lhd Q \oplus V \lhd R$ ($Z\_dres\_rel\_oride\_dist$)
$(Q \oplus R) \rhd V \subseteq Q \rhd V \oplus R \rhd V$ ($Z\_rres\_rel\_oride\_dist$)
If f and g are functions
$(g \oplus f) \cdot x = g \cdot x$ ($Z\_rel\_oride\_beta2$)
$(g \oplus f) \cdot x = f \cdot x$ ($Z\_rel\_oride\_beta1$)

Name

$\_^{+}$	-	Transitive closure
$\_^{*}$	-	Reflexive-transitive closure

Definition

$R^{+} \equiv (\bigcap\ Q \mid Q \in X \leftrightarrow X \land R \subseteq Q \land Q \mathbin{⨾} Q \subseteq Q)$	(*Z_trancl_def*)
$R^{*} \equiv (\bigcap\ Q \mid Q \in X \leftrightarrow X \land \mathrm{id}\ X \subseteq Q \land R \subseteq Q \land Q \mathbin{⨾} Q \subseteq Q)$	(*Z_zrtrancl_def*)

Description

The reflexive and transitive closure operators [5, p 103][2, p 100,101] are treated in the standard HOL distribution, but unfortunately do not accomodate the notion of carrier set for the relation as expected by Z.

Laws

$R \subseteq R^{+}$	(*Z_trancl_inc*)
$R^{+} \mathbin{⨾} R^{+} \subseteq R^{+}$	(*Z_trancl_fcomp_dist*)
$Q \in X \leftrightarrow X \land Q \mathbin{⨾} Q \subseteq Q \land R \subseteq Q \Rightarrow R^{+} \subseteq Q$	(*Z_trancl_subI*)
$\mathrm{id}\ X \subseteq R^{*}$	(*Z_zrtrancl_id*)
$R \subseteq R^{*}$	(*Z_zrtrancl_inc*)
$R^{*} \mathbin{⨾} R^{*} \subseteq R^{*}$	(*Z_zrtrancl_fcomp_dist*)
$\mathrm{id}\ X \subseteq Q \land R \subseteq Q \land Q \mathbin{⨾} Q \subseteq Q \Rightarrow R^{*} \subseteq Q$	(*Z_zrtrancl_subI*)
$R^{*} = R^{+} \cup \mathrm{id}\ X = (R \cup \mathrm{id}\ X)^{+}$	(*Z_zrtrancl_decomp*)
$R^{+} = R \mathbin{⨾} R^{*} = R^{*} \mathbin{⨾} R$	(*Z_trancl_decomp*)
$(R^{+})^{+} = R^{+}$	(*Z_trancl_idem*)
$(R^{*})^{*} = R^{*}$	(*Z_zrtrancl_idem*)
$X \subseteq (R^{*})(\!\vert X \vert\!)$	(*Z_zrtrancl_Image*)
$R(\!\vert (R^{*})(\!\vert X \vert\!) \vert\!) \subseteq (R^{*})(\!\vert X \vert\!)$	(*Z_rel_zrtrancl_Image*)
$U \subseteq V \land R(\!\vert V \vert\!) \subseteq V \Rightarrow (R^{*})(\!\vert U \vert\!) \subseteq V$	(*Z_rel_ztrancl_mono*)

Monotonic Operations

HOL provides a basic monotonicity definition, and we expand upon it to provide the following lemmas.

$$f \in \mathcal{M} \Leftrightarrow (\forall\ S\ T \bullet S \subseteq T \Rightarrow f\,S \subseteq f\,T) \qquad (\mathit{mono\_set\_def})$$

$$\begin{array}{l} f \in \mathcal{M} \wedge \mathit{rev\_args}\ f \in \mathcal{M} \Leftrightarrow \\ (\forall\ S\ T\ U\ V \bullet S \subseteq T \wedge U \subseteq V \Rightarrow f\,S\,U \subseteq f\,T\,V) \end{array} \qquad (\mathit{mono2\_prod\_set\_def})$$

$$f \in \mathcal{M} \Leftrightarrow (\forall\ S\ T \bullet f\,(S \cap T) \subseteq f\,S \cap f\,T) \qquad (\mathit{mono\_set\_inf})$$

$$f \in \mathcal{M} \Leftrightarrow (\forall\ S\ T \bullet f\,S \cup f\,T \subseteq f\,(S \cup T)) \qquad (\mathit{mono\_set\_sup})$$

$$\begin{array}{l} f \in \mathcal{M}_{\sqcup} \wedge \mathit{rev\_args}\ f \in \mathcal{M}_{\sqcup} \Leftrightarrow \\ (\forall\ S\ T\ U\ V \bullet \\ \quad f\,(S \cup T)\,V = f\,S\,V \cup f\,T\,V \wedge \\ \quad f\,S\,(U \cup V) = f\,S\,U \cup f\,S\,V) \end{array} \qquad (\mathit{sup\_morphic2\_set\_def})$$

$$U \lhd (R \cup S) = U \lhd R \cup U \lhd S \qquad (\mathit{dsub\_union\_dist1})$$

$$S \subseteq T \Rightarrow T \lhd R \subseteq S \lhd R \qquad (\mathit{dsub\_mono})$$

$$S = (\bigcap\ T \mid f\,T \subseteq T) \qquad (\mathit{lfp\_set\_def})$$

$$f\,S = S \qquad (\mathit{lfp\_set\_fold})$$

$$\forall\ T \bullet f\,T \subseteq T \Rightarrow S \subseteq T \qquad (\mathit{lfp\_set\_induct})$$

3.4 Functions

Name

$\pfun$	-	Partial functions
$\rightarrow$	-	Total functions
$\pinj$	-	Partial injections
$\rightarrowtail$	-	Total injections
$\psurj$	-	Partial surjections
$\twoheadrightarrow$	-	Total surjections
$\bij$	-	Bijections

Definition

$X \pfun Y \equiv \{ f \mid f \in X \leftrightarrow Y \land (\forall\ x \bullet x \in \operatorname{dom} f \Rightarrow (\exists_1\ y \bullet (x \mapsto y) \in f)) \}$ (*Z_part_funs_def*)
$X \rightarrow Y \equiv \{ f \mid f \in X \pfun Y \land \operatorname{dom} f = X \}$ (*total_funs_def*)
$X \pinj Y \equiv \{ f \mid f \in X \pfun Y \land f^{\sim} \in Y \pfun X \}$ (*part_injs_def*)
$X \rightarrowtail Y \equiv X \pinj Y \cap X \rightarrow Y$ (*total_injs_def*)
$X \psurj Y \equiv \{ f \mid f \in X \pfun Y \land \operatorname{ran} f = Y \}$ (*part_surjs_def*)
$X \twoheadrightarrow Y \equiv X \psurj Y \cap X \rightarrow Y$ (*total_surjs_def*)
$X \bij Y \equiv X \rightarrowtail Y \cap X \twoheadrightarrow Y$ (*bijs_def*)

Description

HOL provides a built-in model for functions (value abstractions), embodied by the type constructor *fun* and the λ-constructor. In the following we refer to this model as the *operator* model of functions.

The single-valued *graphs* provide a convenient (and widely used) mathematical model of functions. This is especially so when partial or finite functions are of particular interest, as is often the case in program specification.

In the following we develop a graph model of functions, based on the Z mathematical toolkit as described by Spivey [5, p 107][2, p 101,102].

Laws

$f \in X \pfun Y \Leftrightarrow f^{\sim} \mathbin{;} f = \operatorname{id}(\operatorname{ran} f)$ (*Z_pfun_left_inv*)
$f \in X \pinj Y \Leftrightarrow f \in X \pfun Y \land f^{\sim} \in Y \pfun X$ (*Z_pinj_f_finv*)
$f \in X \rightarrowtail Y \Leftrightarrow f \in X \rightarrow Y \land f^{\sim} \in Y \pfun X$ (*Z_tinj_f_finv*)
$f(\!|S|\!) \cap f(\!|T|\!) = f(\!|S \cap T|\!)$ (*Z_tinj_image_inter*)
$f \in X \bij Y \Leftrightarrow f \in X \rightarrow Y \land f^{\sim} \in Y \rightarrowtail X$ (*Z_bij_tfun_inv_tinj*)
$f^{\sim} \mathbin{;} f = \operatorname{id} Y$ (*Z_psurj_left_inv*)

Relational operations on functions

Identity relation is a function

$\mathrm{id}\ S \in X \rightarrowtail\!\!\!\!\!\!+\ X$ (*Z_id_pinj*)
$\mathrm{id}\ X \in X \rightarrowtail\!\!\!\!\!\twoheadrightarrow X$ (*Z_id_bij*)
$f \in X \nrightarrow Y \land g \in Y \nrightarrow Z \Rightarrow g \circ f \in X \nrightarrow Z$ (*Z_comp_in_pfunI*)
$f \in X \rightarrow Y \land g \in Y \nrightarrow Z \land \mathrm{ran}\ f \subseteq \mathrm{dom}\ g \Rightarrow g \circ f \in X \rightarrow Z$ (*Z_comp_in_tfunI*)
$f \in X \nrightarrow Y \Rightarrow S \lhd f \in X \nrightarrow Y$ (*Z_dres_in_pfunI*)
$f \in X \nrightarrow Y \Rightarrow f \rhd T \in X \nrightarrow Y$ (*Z_rres_in_pfunI*)
$f \in X \nrightarrow Y \land g \in X \nrightarrow Y \Rightarrow f \oplus g \in X \nrightarrow Y$ (*Z_rel_oride_in_pfunI*)

Composition and restrictions of injections:

$g \circ f \in X \rightarrowtail\!\!\!\!\!\!+\ Z$ (*Z_comp_in_pinjI*)
$f \in X \rightarrowtail\!\!\!\!\!\!+\ Y \Rightarrow S \lhd f \in X \rightarrowtail\!\!\!\!\!\!+\ Y$ (*Z_dres_in_pinjI*)
$f \in X \rightarrowtail\!\!\!\!\!\!+\ Y \Rightarrow f \rhd T \in X \rightarrowtail\!\!\!\!\!\!+\ Y$ (*Z_rres_in_pinjI*)
$f \in X \rightarrowtail\!\!\!\!\!\!+\ Y \Rightarrow f^{\sim} \in Y \rightarrowtail\!\!\!\!\!\!+\ X$ (*Z_pinj_inv_pinj*)

Set theoretic operations

$f \in X \nrightarrow Y \land g \in X \nrightarrow Y \land \mathrm{dom}\ f \cap \mathrm{dom}\ g = \varnothing \Rightarrow f \cup g \in X \nrightarrow Y$ (*Z_union_in_pfun*)
$f \in X \nrightarrow Y \land g \in X \nrightarrow Y \Rightarrow f \cap g \in X \nrightarrow Y$ (*Z_inter_in_pfun*)
$f \in X \rightarrowtail\!\!\!\!\!\!+\ Y \land g \in X \rightarrowtail\!\!\!\!\!\!+\ Y \Rightarrow f \cap g \in X \rightarrowtail\!\!\!\!\!\!+\ Y$ (*Z_inter_in_pinj*)

Special cases

$f \in X \nrightarrow Y \land g \subseteq f \Rightarrow g \in X \nrightarrow Y$ (*Z_subset_in_pfun*)
$f \in X \rightarrowtail\!\!\!\!\!\!+\ Y \land g \subseteq f \Rightarrow g \in X \rightarrowtail\!\!\!\!\!\!+\ Y$ (*Z_subset_in_pinj*)

3.5 Numbers and finiteness

Name

$\mathbb{A}$	-	Numbers
$\mathbb{N}$	-	Natural numbers
$\mathbb{Z}$	-	Integers
$+, -, *,$ div, mod	-	Arithmetic operations
$<, \leq, \geq, >$	-	Numerical comparison

Definition

$\mathbb{N} \equiv (\bigcap N \mid 0 \in N \wedge (\forall\ x \bullet x \in N \Rightarrow x + 1 \in N))$ (*Z_zNats_def*)

$\mathbb{Z} \equiv \{ z \mid z \in \mathbb{A} \wedge (\exists\ x \bullet x \in \mathbb{N} \wedge (z = x \vee z = -\ x)) \}$ (*Z_zInts_def*)

Other definitions not included for brevity.

Description

The number domain [5, p 108][2, p 103] for Z is an abstract set $\mathbb{A}$, pronounced “arithmos”. The basic requirements for arithmos is that it must admit an injective, homomorphic embedding of the integers. Isabelle declares homomorphic embeddings of the naturals and integers, but does not require they be injective. We declare strengthenings of these embeddings and lift natural number and integer lemmas to these embeddings. We omit some definitions of the above operators for brevity.

Laws

$< 0\, b \Rightarrow\ \leq 0\ (a \text{ mod } b) \wedge\ < (a \text{ mod } b)\ b$ (*Z_mod_bounds*)

$b \neq 0 \Rightarrow a = b * (a \text{ div } b) + a \text{ mod } b$ (*Z_div_mod_reconstr*)

$b \neq 0 \wedge c \neq 0 \Rightarrow c * a \text{ div } (c * b) = a \text{ div } b$ (*Z_div_mod_reduce*)

Name

$\mathbb{N}_1$	-	Strictly positive integers
succ	-	Successor function
$(a..b)$	-	Number range

Definition

$\mathbb{N}_1 \equiv \mathbb{N} \setminus \{0\}$ (*zNats1_def*)

$\mathrm{succ} \equiv (\lambda\ n \mid n \in \mathbb{N} \bullet n + 1)$ (*Z_zsucc_def*)

$a..b \equiv \{\ k \mid k \in \mathbb{Z} \land a \leq k \leq b\ \}$ (*Z_zint_range_def*)

Description

The non-zero integers [5, p 109][2, p 105] are not defined in HOL. We introduce them as a subset of arithmos.

The successor [5, p 109][2, p 103] is defined as an the operator in HOL, but there is a strong assumption in Z that the successor is a graph. Hence we introduce a graph-style successor.

Numeric ranges [5, p 109][2, p 106] are defined in HOL, but we find it more convenient to introduce a Z specific range.

Laws

$\mathrm{succ} \in \mathbb{N} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbb{N}_1$ (*zsucc_bij*)

$\forall\ n \bullet n \in \mathbb{N}\ \Rightarrow \mathrm{succ}{\cdot}n = n + 1$ (*Z_zsucc_beta*)

$<\ m\ n\ \Rightarrow n..m = \varnothing$ (*Z_zint_range_empty*)

$a..a = \{a\}$ (*zint_range_singleton*)

$n_1..m_1 \subseteq n_2..m_2$ (*zint_range_mono*)

Name

$R^{n}[X]$ - Iteration

Definition

$R^{0} \equiv \mathrm{id}\ X$	(*Z_ziter_zero_def*)
$R^{n+1} \equiv R \mathbin{;} R^{n}$	(*Z_ziter_iter_def*)
$R^{-k} \equiv (R^{\sim})^{k}$	(*Z_ziter_minus_k_def*)

Description

HOL already defines a relational iteration operator [5, p 110][2, p 106], but, as per the transitive closure operator, it does not take account of a carrier set as the Z operator does nor is it defined over the full integer space. Thus we are forced to redefine a Z compliant version of iteration.

Laws

$R^{0} = \mathrm{id}\ X$	(*Z_ziter_zero*)
$R^{1} = R$	(*Z_ziter_one*)
$R^{2} = R \mathbin{;} R$	(*Z_ziter_two*)
$R^{-1} = R^{\sim}$	(*Z_ziter_minus_one*)
$R^{n+1} = R \mathbin{;} R^{n}$	(*Z_ziter_iter*)
$R^{n+1} = R^{n} \mathbin{;} R$	(*Z_ziter_iter′*)
$(R^{\sim})^{n} = (R^{n})^{\sim}$	(*Z_ziter_converse*)
$R^{n+m} = R^{n} \mathbin{;} R^{m}$	(*Z_ziter_add_dist*)
$R^{n * m} = (R^{n})^{m}$	(*Z_ziter_mult_dist*)
$R^{+} = (\bigcup\ k \mid \leq (1{::}\beta)\ k \land k \in \mathbb{Z} \bullet R^{k})$	(*Z_ziter_ztrancl*)
$R^{*} = (\bigcup\ k \mid \leq (0{::}\beta)\ k \land k \in \mathbb{Z} \bullet R^{k})$	(*Z_ziter_zrtrancl*)
$R \mathbin{;} S = S \mathbin{;} R \Rightarrow (R \mathbin{;} S)^{k} = R^{k} \mathbin{;} S^{k}$	(*Z_fcomp_ziter*)

Name

$\mathbb{F}$	-	Finite sets
$\mathbb{F}_1$	-	Non-empty finite sets
$\#\_$	-	Number of members of a set

Definition

$\mathbb{F}\ X \equiv \{\ S \mid S \in \mathbb{P}\ X \land (\exists\ n \bullet n \in \mathbb{N} \land (\exists\ f \bullet f \in (1..n) \rightarrowtail\!\!\!\!\twoheadrightarrow S))\ \}$ (*Z_fin_pow_def*)

$\mathbb{F}_1\ X \equiv \mathbb{F}\ X \setminus \{\varnothing\}$ (*Z_fin_pow1_def*)

$\#S \equiv (\mu\ n \mid n \in \mathbb{N} \land (\exists\ f \bullet f \in (1..n) \rightarrowtail\!\!\!\!\twoheadrightarrow S))$ (*Z_zcard_def*)

Description

We introduce finite subsets and finite non-empty subsets [5, p 111][2, p 97] as set operators. We define them as restrictions of the existing HOL finite set operator.

Laws

$S \in \mathbb{F}\ X \Leftrightarrow (\forall\ f \bullet f \in S \rightarrowtail S \Rightarrow \mathrm{ran}\ f = S)$ (*Z_finite_iff*)

$\varnothing \in \mathbb{F}\ X$ (*Z_empty_fin_pow*)

$\forall\ S\ x \bullet S \in \mathbb{F}\ X \land x \in X \Rightarrow S \cup \{x\} \in \mathbb{F}\ X$ (*Z_fin_pow_insert*)

$\#(S \cup T) = \#S + \#T - \#(S \cap T)$ (*Z_zcard_union*)

$\mathbb{F}_1\ X = \{\ S \mid S \in \mathbb{F}\ X \land <\ 0\ (\#S)\ \}$ (*Z_fin_pow1_redef*)

Name

$\mathrel{+\!\!\!\!\!\twoheadrightarrow}$	-	Finite partial functions
$\rightarrowtail\!\!\!\!\!\!+\!\!\!\!\twoheadrightarrow$	-	Finite partial injections

Definition

$X \mathrel{+\!\!\!\!\!\twoheadrightarrow} Y \equiv \{ f \mid f \in X \mathrel{+\!\!\!\!\!\rightarrow} Y \land \mathrm{dom}\, f \in \mathbb{F}\, X \}$ (*Z_finite_part_funs_def*)

$X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\twoheadrightarrow Y \equiv X \mathrel{+\!\!\!\!\!\twoheadrightarrow} Y \cap X \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\rightarrow Y$ (*finite_part_injs*)

Description

Finite functions [5, p 112][2, p 102] are those represented by a finite set of maplets.

Laws

$X \mathrel{+\!\!\!\!\!\twoheadrightarrow} Y = X \mathrel{+\!\!\!\!\!\rightarrow} Y \cap \mathbb{F}\,(X \times Y)$ (*Z_finite_part_fun_fpow*)

Name

min, max - Minimum and maximum of a set of numbers

Definition

$\min \equiv \{\ S\ m \mid S \in \mathbb{P}_1\ \mathbb{Z} \land m \in \mathbb{Z} \land m \in S \land (\forall\ n \bullet n \in S\ \Rightarrow\ \leq m\ n) \bullet S \mapsto m\ \}$ (*zmin_def*)

$\max \equiv \{\ S\ m \mid S \in \mathbb{P}_1\ \mathbb{Z} \land m \in \mathbb{Z} \land m \in S \land (\forall\ n \bullet n \in S\ \Rightarrow\ m \geq n) \bullet S \mapsto m\ \}$ (*zmax_def*)

Description

The minimum and maximum [5, p 113][2, p 107] of a finite set are defined generally. Such functions are defined in HOL, but we introduce graph-based versions in support of Z.

Laws

$\mathbb{F}_1\ \mathbb{Z} \subseteq \mathrm{dom}\ \min$	(*Z_fin_pow1_dom_min*)
$\mathbb{F}_1\ \mathbb{Z} \subseteq \mathrm{dom}\ \max$	(*Z_fin_pow1_dom_max*)
$\mathbb{P}\ \mathbb{N} \cap \mathrm{dom}\ \min = \mathbb{P}_1\ \mathbb{N}$	(*Z_pow_zmin_pow1*)
$\mathbb{P}\ \mathbb{N} \cap \mathrm{dom}\ \max = \mathbb{F}_1\ \mathbb{N}$	(*Z_pow_zmax_fpow1*)
$\min\cdot(S \cup T) = \min\cdot\{\min\cdot S, \min\cdot T\}$	(*Z_min_union*)
$\max\cdot(S \cup T) = \max\cdot\{\max\cdot S, \max\cdot T\}$	(*Z_max_union*)
$\min\cdot(S \cap T) \geq \min\cdot S$	(*Z_min_inter*)
$\leq (\max\cdot(S \cap T))\ (\max\cdot S)$	(*Z_max_inter*)
$\leq a\ b\ \Rightarrow \min\cdot(a..b) = a \land \max\cdot(a..b) = b$	(*Z_min_max_zint_range*)
$(a..b) \cap (c..d) = \max\cdot\{a, c\}..\min\cdot\{b, d\}$	(*Z_zint_range_inter_min_max*)

Proof by induction

Arithmetic induction provides a method for proving a number of theorems about the natural numbers.

zNats_induct:

$$[\![n \in \mathbb{N}; P\ 0; \bigwedge m \bullet [\![m \in \mathbb{N}; P\ m]\!] \vdash P\ (m + 1)]\!] \vdash P\ n$$

3.6 Sequences

Name

seq	-	Finite sequences
seq_1	-	Non-empty finite sequences
iseq	-	Injective sequences
sinsert	-	Sequence insertion
$\langle\,\rangle$	-	Empty sequence

Definition

$\mathrm{seq}\ X \equiv (\bigcup\ n \mid n \in \mathbb{N} \bullet (1..n) \to X)$ (*seq_def*)

$\mathrm{seq}_1\ X \equiv \{\ s \mid s \in \mathrm{seq}\ X \wedge <0\ (\#s)\ \}$ (*seq1_def*)

$\mathrm{iseq}\ X \equiv \mathrm{seq}\ X \cap \mathbb{N} \rightarrowtail\!\!\!\!\!\!\!\!\;\;\;\; \mapsto X$ (*iseq_def*)

$\mathit{sinsert}\ x\ s \equiv \{(1, x)\} \oplus \{\ n\ x \mid (n, x) \in s \bullet (n + 1, x)\ \}$ (*sinsert_def*)

$\langle\,\rangle \equiv \varnothing$ (*sempty_def*)

Notation

We write $\langle x_0, x_1, \ldots, x_n\rangle$ for the sequence $\mathit{sinsert}\ x_0\ \langle x_1, \ldots\ x_n\rangle$.

Description

HOL includes an extensive theory of lists, but the Z notion of sequences [5, p 115][2, p 107] modelled as graphs are not part of HOL. We develop a syntax and type constructors for graph-based sequences; building upon the function and number theories discussed previously. A basic sequence is a finite graph defined on an initial interval of the natural numbers. A non-empty sequence has at least one element and an injective sequence has no repeated elements.

Laws

$\mathrm{seq}_1\ X = \mathrm{seq}\ X \setminus \{\langle\,\rangle\}$ (*seq1_nonempty*)

Name

$\frown$	-	Concatenation
rev	-	Reverse

Definition

$s \frown t \equiv s \cup \{ n \mid n \in \mathrm{dom}\, t \bullet n + \#s \mapsto t \cdot n \}$ (*Z_sconcat_redef*)

$\mathrm{rev}\, s \equiv (\lambda\, n \mid n \in \mathrm{dom}\, s \bullet s \cdot (\#s - n + 1))$ (*Z_srev_def*)

Description

Sequence concatenation [5, p 116][2, p 108] adds the elements of one sequence at the end of another.

Sequence reverse [5, p 116][2, p 108] maintains the elements of its argument, listing them in the reverse order.

Laws

$s \frown t \frown u = s \frown (t \frown u)$ (*Z_sconcat_assoc*)

$\langle\,\rangle \frown s = s$ (*Z_sconcat_semptyl*)

$s \frown \langle\,\rangle = s$ (*Z_sconcat_semptyr*)

$\#(s \frown t) = \#s + \#t$ (*Z_sconcat_zcard*)

$\mathrm{rev}\, \langle\,\rangle = \langle\,\rangle$ (*Z_srev_sempty*)

$\mathrm{rev}\, \langle x \rangle = \langle x \rangle$ (*Z_srev_sunit*)

$\mathrm{rev}\, (s \frown t) = \mathrm{rev}\, t \frown \mathrm{rev}\, s$ (*Z_srev_sconcat*)

$\mathrm{rev}\, (\mathrm{rev}\, s) = s$ (*Z_srev_srev*)

Name

$head, tail, last, front$ - Sequence decomposition

Definition

$head\ s \equiv s \cdot 1$	(Z_shead_def)
$tail\ s \equiv (\lambda\ n \mid n \in 1..\#s - 1 \bullet s \cdot (n + 1))$	(Z_stail_def)
$front\ s \equiv (1..\#s - 1) \lhd s$	(Z_sfront_def)
$last\ s \equiv s \cdot \#s$	(Z_slast_def)

Description

The head, tail, front and last operators [5, p 117][2, p 108,9] are defined as in Spivey.

Laws

$head\ \langle x \rangle = last\ \langle x \rangle = x$	($Z\_shead\_slast\_sunit$)
$tail\ \langle x \rangle = front\ \langle x \rangle = \langle\ \rangle$	($Z\_stail\_sfront\_sunit$)
$s \neq \langle\ \rangle \Rightarrow head\ (s \frown t) = head\ s \land tail\ (s \frown t) = tail\ s \frown t$	($Z\_shead\_stail\_sconcat$)
$t \neq \langle\ \rangle \Rightarrow last\ (s \frown t) = last\ t \land front\ (s \frown t) = s \frown front\ t$	($Z\_slast\_sfront\_sconcat$)
$s \neq \langle\ \rangle \Rightarrow \langle head\ s \rangle \frown tail\ s = s$	($Z\_shead\_stail\_reconstr$)
$s \neq \langle\ \rangle \Rightarrow front\ s \frown \langle last\ s \rangle = s$	($Z\_sfront\_slast\_reconstr$)
$s \neq \langle\ \rangle \Rightarrow head\ (\mathrm{rev}\ s) = last\ s \land tail\ (\mathrm{rev}\ s) = \mathrm{rev}\ (front\ s)$	($Z\_shead\_stail\_srev\_sfront$)
$s \neq \langle\ \rangle \Rightarrow last\ (\mathrm{rev}\ s) = head\ s \land front\ (\mathrm{rev}\ s) = \mathrm{rev}\ (tail\ s)$	($Z\_slast\_sfront\_srev$)

Name

$\upharpoonleft$	-	Extraction
$\upharpoonright$	-	Filtering
squash	-	Compaction

Definition

$U \upharpoonleft s \equiv squash\ (U \lhd s)$ ($Z\_sxtract\_def$)

$s \upharpoonright V \equiv squash\ (s \rhd V)$ ($Z\_sfilter\_def$)

$squash\ f \equiv \{\ x \mid x \in \mathrm{dom}\ f \bullet \#\{\ i \mid i \in \mathrm{dom}\ f \land <\ i\ x\ \} + 1 \mapsto f \cdot x\ \}$ ($ssquash\_def$)

Description

We can create a sequence *squash f* from a function, by translating its domain using the *bounded_card* function. The inverse of this function is used to show monotonicity of the squash function [5, p 118][2, p 109]. Extraction and filtering [5, p 118][2, p 109] are defined in the natural way.

Laws

$\langle\ \rangle \upharpoonright V = U \upharpoonleft \langle\ \rangle = \langle\ \rangle$ ($Z\_sfilter\_sxtract\_sempty$)

$s \frown t \upharpoonright V = (s \upharpoonright V) \frown (t \upharpoonright V)$ ($Z\_sconcat\_sfilter$)

$\mathrm{ran}\ s \subseteq V \Leftrightarrow s \upharpoonright V = s$ ($Z\_sfilter\_ran\_redef$)

$s \upharpoonright \varnothing = \varnothing \upharpoonleft s = \langle\ \rangle$ ($Z\_sfilter\_empty\_sxtract\_sempty$)

$\leq (\#(s \upharpoonright V))\ (\#s)$ ($Z\_zcard\_sfilter$)

$s \upharpoonright V \upharpoonright W = s \upharpoonright (V \cap W)$ ($Z\_sfilter\_repeat$)

Name

prefix	-	Prefix relation
suffix	-	Suffix relation
in	-	Segment relation

Definition

$s \;\mathsf{prefix}\; t \equiv \exists\; v \bullet v \in \mathrm{seq}\, X \wedge s \frown v = t$ (*Z_prefix_def*)

$s \;\mathsf{suffix}\; t \equiv \exists\; u \bullet u \in \mathrm{seq}\, X \wedge u \frown s = t$ (*Z_suffix_def*)

$s \;\mathsf{in}\; t \equiv \exists\; u\; v \bullet (u \in \mathrm{seq}\, X \wedge v \in \mathrm{seq}\, X) \wedge u \frown s \frown v = t$ (*Z_infix_def*)

Description

Prefix, suffix and infix [5, p 119][2, p 109,110] are defined as type generic operators. The extraction lemmas discussed below are not expressed as in Spivey. All three require the extra assumption that $\#s \leq \#t$ to establish various arithmetic expressions. This must be an unstated assumption in the definition of suffix, prefix and infix since if s is larger than t it obviously cannot be a suffix, prefix or infix of t anyway.

The infix extraction lemma requires a complete restatement as compared to the version in Spivey. The original expression in Spivey is:

$$s \;\mathsf{in}\; t \Leftrightarrow (\exists\; n \mid n \in \{1..\#t\} \bullet s = \{n..n + \#s\} \upharpoonleft t)$$

which is clearly wrong since $\#\{n..n + \#s\} > \#s$.

Laws

$s \;\mathsf{prefix}\; t \Leftrightarrow s = (1..\#s) \upharpoonleft t$ (*Z_sprefix_sxtract*)

$s \;\mathsf{suffix}\; t \Leftrightarrow s = (\#t - \#s + 1..\#t) \upharpoonleft t$ (*Z_ssuffix_sxtract*)

$s \;\mathsf{in}\; t \Leftrightarrow (\exists\; n \bullet n \in 0..\#t - \#s \wedge s = (n + 1..n + \#s) \upharpoonleft t)$ (*Z_sinfix_sxtract*)

$s \;\mathsf{in}\; t \Leftrightarrow (\exists\; u \bullet u \in \mathrm{seq}\, X \wedge s \;\mathsf{suffix}\; u \wedge u \;\mathsf{prefix}\; t)$ (*Z_sinfix_sp*)

$s \;\mathsf{in}\; t \Leftrightarrow (\exists\; v \bullet v \in \mathrm{seq}\, X \wedge s \;\mathsf{prefix}\; v \wedge v \;\mathsf{suffix}\; t)$ (*Z_sinfix_ps*)

Relational operations on sequences

As in Spivey [5, p 120], our definition of sequence makes it a special type of graph, and many previously discussed operators are applicable to sequences.

$\#(f \circ s) = \#s$	$(\mathit{Z\_seq\_rel\_comp\_zcard})$
$\forall\ i \bullet i \in 1..\#s \Rightarrow (f \circ s) \cdot i = f \cdot s \cdot i$	$(\mathit{Z\_seq\_rel\_comp\_beta})$
$f \circ \langle\ \rangle = \langle\ \rangle$	$(\mathit{Z\_sempty\_rel\_comp})$
$f \circ \langle x \rangle = \langle f \cdot x \rangle$	$(\mathit{Z\_sunit\_rel\_comp})$
$f \circ s \frown t = (f \circ s) \frown (f \circ t)$	$(\mathit{Z\_sconcat\_rel\_comp})$
$\mathrm{ran}\ s = \{\ i \mid i \in 1..\#s \bullet s \cdot i\ \}$	$(\mathit{Z\_ran\_redef})$
$\mathrm{ran}\ \langle\ \rangle = \varnothing$	$(\mathit{Z\_sempty\_ran})$
$\mathrm{ran}\ \langle x \rangle = \{x\}$	$(\mathit{Z\_sunit\_ran})$
$\mathrm{ran}\ (s \frown t) = \mathrm{ran}\ s \cup \mathrm{ran}\ t$	$(\mathit{Z\_sconcat\_ran\_union})$
$\mathrm{rev}\ (f \circ s) = f \circ \mathrm{rev}\ s$	$(\mathit{Z\_srev\_rel\_comp})$
$(f \circ s) \upharpoonright V = f \circ s \upharpoonright (f^{\sim})(\!\vert V \vert\!)$	$(\mathit{Z\_sfilter\_rel\_comp})$
$\mathrm{ran}\ (s \upharpoonright V) = \mathrm{ran}\ s \cap V$	$(\mathit{Z\_ran\_sfilter\_inter})$

Name

$\frown/$ - Distributed concatenation

Definition

$\frown/ \langle\,\rangle \equiv \langle\,\rangle$ (*sdistrib_sempty*)
$\frown/ \langle s \rangle \equiv s$ (*Z_sdistrib_sunit*)
$\frown/ (s \frown t) \equiv (\frown/ s) \frown (\frown/ t)$ (*Z_sdistrib_sconcat*)

Description

Distributed concatenation [5, p 121][2, p 110] is modelled as a type generic operator built from a recursion operator. The recursion operator allows for partial evaluation of functions over a sequence.

Laws

$\frown/ \langle s, t \rangle = s \frown t$ (*Z_sdistrib_sinsert_seq*)
$\mathrm{rev}\,(\frown/ q) = \frown/ \mathrm{rev}\,(\mathrm{grf}\ \mathrm{rev} \circ q)$ (*Z_srev_sdistrib*)
$\frown/ q \upharpoonright V = \frown/ ((\lambda\, s \mid s \in \mathrm{seq}\, X \bullet s \upharpoonright V) \circ q)$ (*Z_sdistrib_sfilter*)
$f \circ \frown/ q = \frown/ ((\lambda\, s \mid s \in \mathrm{seq}\, X \bullet f \circ s) \circ q)$ (*Z_sdistrib_rel_comp*)
$\mathrm{ran}\,(\frown/ q) = (\bigcup\ i \mid i \in 1..\#q \bullet \mathrm{ran}\,(q \cdot i)) = \bigcup \mathrm{ran}\,(\mathrm{grf}\ \mathrm{ran} \circ q)$ (*Z_sdistrib_ran*)

Name

disjoint	-	Disjointness
partition	-	Partitions

Definition

disjoint $S \equiv \forall\ i\ j \bullet i, j \in \mathrm{dom}\ S \land i \neq j \ \Rightarrow S{\cdot}i \cap S{\cdot}j = \varnothing$ (*Z_Disjoint_def*)

S partition $T \equiv$ disjoint $S \land (\bigcup\ i \mid i \in \mathrm{dom}\ S \bullet S{\cdot}i) = T$ (*Z_zpartition_def*)

Description

We define type generic operators for disjoint and partition [5, p 122]. The disjoint operator was defined in HOL, we have simply provided an alternate definition more in keeping with the Z definition.

Laws

disjoint $\varnothing$ (*Z_Disjoint_empty*)

disjoint $\langle x \rangle$ (*Z_Disjoint_sunit*)

disjoint $\langle A, B \rangle \Leftrightarrow A \cap B = \varnothing$ (*Z_Disjoint_sinsert*)

$\langle A, B \rangle$ partition $C \Leftrightarrow A \cap B = \varnothing \land A \cup B = C$ (*Z_partition_sinsert*)

Induction

We provide three induction methods for sequences [5, p 123]. The first involves an insertion at the head of a sequence, the second involves an insertion at the end of the sequence and the third involves the concatenation of two sequences.

seq_induct:

$$[\![s \in \text{seq}\ X;\ P\ \langle\ \rangle;\ \bigwedge\ xs\ x \bullet [\![xs \in \text{seq}\ X;\ x \in X;\ P\ xs]\!] \vdash P\ (\langle x \rangle \mathbin{^\frown} xs)]\!] \vdash P\ s$$

seq_rev_induct:

$$[\![s \in \text{seq}\ X;\ P\ \langle\ \rangle;\ \bigwedge\ xs\ x \bullet [\![xs \in \text{seq}\ X;\ x \in X;\ P\ xs]\!] \vdash P\ (xs \mathbin{^\frown} \langle x \rangle)]\!] \vdash P\ s$$

seq_sconcat_induct:

$$[\![s \in \text{seq}\ X;\ P\ \langle\ \rangle;\ \bigwedge\ x \bullet x \in X\ \vdash P\ \langle x \rangle;\ \bigwedge\ s\ t \bullet [\![s \in \text{seq}\ X;\ t \in \text{seq}\ X;\ P\ s;\ P\ t]\!] \vdash P\ (s \mathbin{^\frown} t)]\!] \vdash P\ s$$

3.7 Bags

Name

bag	-	Bags
$\sharp$	-	Multiplicity
$\otimes$	-	Bag scaling
binsert	-	Bag insertion
$\llbracket\,\rrbracket$	-	Empty bag

Definition

$\mathrm{bag}\ X \equiv X \nrightarrow \mathbb{N}_1$ (*bag_def*)

$b \sharp x \equiv ((\lambda\, x \bullet 0) \oplus b)\ x$ (*Z_bhash_def*)

$(n \otimes b) \sharp x \equiv n * b \sharp x$ (*Z_bscale_def*)

$\mathit{binsert}\ x\ b \equiv b \oplus \{(x, b \sharp x + 1)\}$ (*binsert_def*)

$\llbracket\,\rrbracket \equiv \varnothing$ (*bempty_def*)

Notation

We write $\llbracket x_0, x_1, \ldots, x_n \rrbracket$ for $\mathit{binsert}\ x_0\ \llbracket x_1, \ldots, x_n \rrbracket$.

Description

Bags are collections that may be distinguished by the number of occurrrences of a member. This makes them essentially natural number valued functions. Spivey [5, p. 124] introduces bags as non-zero valued functions from a specified range set to the natural numbers.

Laws

$\mathrm{dom}\ b = \{\ x \mid b \sharp x \in \mathbb{N}_1\ \}$ (*Z_dom_bhash*)

$n \otimes \llbracket\,\rrbracket = 0 \otimes b = \llbracket\,\rrbracket$ (*Z_bscale_bempty_zero*)

$1 \otimes b = b$ (*Z_unit_scale*)

$(n * m) \otimes b = n \otimes m \otimes b$ (*Z_dist_scale*)

Name

$\sqsubset$	-	Bag membership
$\sqsubseteq$	-	Sub-bag relation

Definition

$x \sqsubset B \equiv x \in \mathrm{dom}\ B$ (*Z_inbag_def*)

$B \sqsubseteq C \equiv \forall\ x \bullet x \in X \Rightarrow \leq (B \sharp x)\ (C \sharp x)$ (*Z_bag_le_def*)

Description

The bag membership operator [5, p. 125] determines the frequency of a particular element is non-zero.

The sub-bag operator [5, p. 125] is the point-wise lift of the natural number order.

Laws

$x \sqsubset b \Leftrightarrow <\ 0\ (b \sharp x)$ (*Z_inbag_bhash*)

$b \sqsubseteq c \Rightarrow \mathrm{dom}\ b \subseteq \mathrm{dom}\ c$ (*Z_bag_le_dom*)

$[\![\ ]\!] \sqsubseteq b$ (*Z_bempty_bag_le*)

$b \sqsubseteq b$ (*Z_bag_self_le*)

$b \sqsubseteq c \wedge c \sqsubseteq b \Rightarrow b = c$ (*Z_bag_le_eq*)

$b \sqsubseteq c \wedge c \sqsubseteq d \Rightarrow b \sqsubseteq d$ (*Z_bag_le_trans*)

Name

$\uplus$	-	Bag union
$\uminus$	-	Bag difference

Definition

$(b \uplus c) \sharp x \equiv b \sharp x + c \sharp x$ (*Z_bunion_def*)

$(b \uminus c) \sharp x \equiv \text{if} \leq (c \sharp x)\ (b \sharp x) \quad \text{then } b \sharp x - c \sharp x \quad \text{else } \mathit{0} \text{ fi}$ (*Z_bdiff_def*)

Description

Bag union and bag difference [5, p. 126] can be defined in terms of arithmetic on the counts.

Laws

$\text{dom}\ (b \uplus c) = \text{dom}\ b \cup \text{dom}\ c$ (*Z_bunion_dom*)

$[\![\ ]\!] \uplus b = b \uplus [\![\ ]\!] = b$ (*Z_bunion_empty*)

$b \uplus c = c \uplus b$ (*Z_bunion_commute*)

$b \uplus c \uplus d = b \uplus (c \uplus d)$ (*Z_bunion_assoc*)

$[\![\ ]\!] \uminus b = [\![\ ]\!] \land b \uminus [\![\ ]\!] = b$ (*Z_bdiff_empty*)

$b \uplus c \uminus c = b$ (*Z_bunion_inverse*)

$(n + m) \otimes b = n \otimes b \uplus (m \otimes b)$ (*Z_bscale_union*)

$\leq m\ n \Rightarrow (n - m) \otimes b = n \otimes b \uminus (m \otimes b)$ (*Z_bscale_diff*)

$n \otimes b \uplus c = n \otimes b \uplus (n \otimes c)$ (*Z_bunion_distr*)

$n \otimes b \uminus c = n \otimes b \uminus (n \otimes c)$ (*Z_bdiff_distr*)

Name

items - Bag of elements of a sequence

Definition

(items s) $\sharp$ $x \equiv \#\{ i \mid i \in \mathrm{dom}\ s \wedge s{\cdot}i = x \}$ $\quad$ (*Z_bitems_def*)

Description

A bag can be constructed by counting the occurrences of the elements of a list [5, p. 127].

Laws

dom (items s) = ran s $\quad$ (*Z_bitems_dom*)

items (*sinsert* x s) = *binsert* x (items s) $\quad$ (*Z_bitems_sinsert*)

items ($s \frown t$) = items $s \uplus$ items t $\quad$ (*Z_bitems_concat*)

items s = items $t \Leftrightarrow (\exists\ f \bullet f \in \mathrm{dom}\ s \rightarrowtail\!\!\!\twoheadrightarrow \mathrm{dom}\ t \wedge s = t \circ f)$ $\quad$ (*Z_bitems_permutations*)

References

1. I. J. Hayes, editor. *Specification Case Studies*. Prentice Hall International, second edition, 1993.
2. International Organization for Standardization. *Information technology – Z formal specification notation – Syntax, type system and semantics*, 2002.
3. B. Mahony, J. McCarthy, K. Williams, and Linh Vu. The HiVe mathematical toolkit, part 2: The Z mathematical toolkit. To be published as DSTO General Document (~350 pages), May 2008.
4. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
5. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International, second edition, 1992.

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. CAVEAT/PRIVACY MARKING

2. TITLE Z Support in the HiVe Mathematical Toolkit (U)	3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)
4. AUTHORS Brendan Mahony, Jim McCarthy, Linh Vu, Kylie Williams	5. CORPORATE AUTHOR Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia

6a. DSTO NUMBER DSTO–TR–2272	6b. AR NUMBER AR–014–433	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE March, 2009

8. FILE NUMBER	9. TASK NUMBER DMO 07/007	10. SPONSOR DMO	11. No OF PAGES 59	12. No OF REFS 5

13. URL OF ELECTRONIC VERSION http://www.dsto.defence.gov.au/corporate/reports/DSTO–TR–2272.pdf	14. RELEASE AUTHORITY Chief, Command, Control, Communications and Intelligence Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved For Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS

No Limitations

18. DSTO RESEARCH LIBRARY THESAURUS

Software engineering
Requirements management
Standards

19. ABSTRACT (U)

The HiVe project is an ambitious research programme aimed at providing DSTO and the Australian Defence Department with the world's most advanced assurance tools. A key part of this is the provision of advanced high assurance analysis tools in the form of the HiVe Modeller component.
Formal specification and system modelling activities in the HiVe Modeller are supported through an Isabelle/HOL implementation of the HiVe Mathematical Toolkit. This report describes support for the Z Mathematical Toolkit within the HiVe Mathematical Toolkit.